1) Practical view: a tiny operator that gives a whole transform family

Definition (one pass): for $n = 2^m$,

$$S = P\,(I \otimes H_2), \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

i.e. pairwise $(a, b) \mapsto (a + b, a - b)$, then pack all sums in the first half and all differences in the second.

Key facts

- Normalize with $1/\sqrt{2} \to \tilde{S}$ is orthogonal (energy preserving).
- Repeat the *same pass* $m = \log_2 n$ times → you get the Walsh–Hadamard transform (up to ordering).
- Keep iterating → exact cycle (period $2m$ for $\tilde{S}$). Every power S^k is a different, useful mixer.

What it *does*

- Each step = local **average/difference** + global **reshuffle**.
- Iteration = a **walk on the hypersphere** (after normalization): structured rotations that go from local → global mixing → back.
- You get a **family of transforms** $\{S^k\}$: early k = cheap partial mixing; $k = m$ = full Hadamard.

Two ultra-cheap "filters"

- $(I + S)x$: keeps components that **change slowly** under one step (low-pass-ish).
- $(I - S)x$: keeps components that **change quickly** (high-pass-ish).
- Better filters come from polynomials $p(S) = \sum a_j S^j$ (see below).

Why it's attractive

- Only adds/subtracts + permutation (no multiplies).
- Same kernel each pass → easy SIMD/GPU/streaming.
- You can stop early (tunable cost vs. mixing).

Java Code

```
// x -> y = S x (unnormalized)
void applyS(double[] x, double[] y) {
   int n = x.length, h = n >> 1;
   for (int i = 0; i < h; i++) {
      double a = x[2*i], b = x[2*i+1];
      y[i]   = a + b;    // sums
      y[i+h]  = a - b;    // differences
   }
}
```

2) Technical notes, pointers, and keywords

- **Operator form**
  - $\tilde{S} = \frac{1}{\sqrt{2}} P(I \otimes H_2) \in O(n)$, finite order: $\tilde{S}^{2m} = I$.
  - Algebra: $\mathbb{R}[S]/(S^{2m} - I)$.
- **Invariant subspaces**
  - Not independent 2-point blocks: permutation couples them.
  - Decomposition: $\mathbb{R}^n = \bigoplus V_j$, each V_j is 1D (±1) or 2D rotation plane.
  - On V_j: $\tilde{S}|_{V_j} = R(\theta_j)$ with eigenvalues $e^{\pm i\theta_j}$.
  - Intuition: **coupled oscillators on permutation cycles →** eigenvectors are **Fourier modes on cycles** (global, not local).
- **Spectral probing without eigendecomposition**
  - Iterate $x_{k+1} = \tilde{S}x_k$; measure $y_k = \langle v, x_k \rangle$.
  - DFT of y_k → peaks at θ_j (operator's "frequencies").
  - Alternatives: autocorrelation, multi-probe averaging, spectral entropy.

- **Filtering via polynomials (DSP-on-S)**
  - $p(S)x = \sum a_j S^j x$ with response $p(e^{i\theta})$.
  - Examples:
    - Low-pass: $p(S) = \frac{1}{K} \sum_{j=0}^{K-1} S^j$
    - High-pass: $(I - S)^k$
    - Band-pass: $\sum \cos(\omega j) S^j$
  - Computation = repeated S applications (no matrices).
  - Viewpoint aligns with graph signal processing (S as a shift).
- **Krylov subspaces**
  - $\mathcal{K}_k(x) = \text{span}\{x, Sx, \ldots, S^{k-1}x\}$.
  - All $p(S)x$ live in $\mathcal{K}_k$; small k captures dominant modes.
- **Connections / keywords**
  - Butterfly networks, bit permutations, Walsh functions, circulant/cyclic actions.
  - Chebyshev polynomials (sharp spectral filters), Lanczos/power methods.
  - Multiresolution / Haar-like splitting, preconditioning, randomized embeddings.

- **Metrics to monitor**
  - Energy entropy $H = -\sum p_i \log p_i$ (mixing).
  - Correlation $C(k) = \langle x_0, x_k \rangle / \|x_0\|^2$ (decay/periodicity).
  - Spectral entropy of y_k's DFT (structure vs. randomness).
  - Conditioning (unnormalized growth), pairwise decorrelation.
- **Practical patterns**
  - Early-stop S^k for cheap mixing; full $k = m$ for exact Hadamard.
  - Use $(I \pm S)$ or $p(S)$ as fixed, zero-multiply layers (DSP/ML).
  - SDSD... (with diagonal sign flips) for richer, still cheap transforms.

One-line takeaway:

Treat S as a shift; polynomials in S give a complete, multiply-free filtering toolkit over its rotation modes, with the full Hadamard transform appearing at a special iterate.